Subject: Re: semantics for namespace naming
Posted by serue on Thu, 14 Dec 2006 15:36:31 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> Serge E. Hallyn wrote:
> > Let's say we have a vserver, from which we start some jobs
> > which we want to checkpoint/restart/migrate. These are two
> > of the usages we currently foresee for the namespaces, though
> > I'd say it's safe to assume there will be more.
> >
> > I'll want to be able to address the c/r jobs by some ID in
> > order to checkpoint and kill them. I'll also want to be
> > able to address the entire vserver by some ID, in order to
> > kill it. In that case the c/r jobs should also be killed.
> > So those jobs are known by at least two id's. Furthermore, I
> > may want two vservers on the same machine, both running a c/r
> > job called 'calculate_pi'.
> >
> > So we can look at this as a filesystem. In the above scenario,
> > we've got /sergesvserver, /sergesvserver/calculate_pi,
> > /randomvserver, and /randomvserver/calculate_pi. And, if
> > user hallyn logs into /sergesvserver using pam_namespace.so,
> > unsharing his mounts namespace to get a private /tmp and /home,
> > then he ends up in /sergesvserver/unnamed1. So each nsproxy
> > has a node in the namespace id filesystem, with random names
> > unless/until it is renamed to a more meaningful name. This
> > allows us to switch to a vserver by specifying the vserver's
> > name (ln /sys/namespaces/vserver1 /proc/nsproxy or whatever
> > semantics we end up using), kill an entire vserver recursively
> > (rm -rf /sys/namespaces/vserver1), perhaps even checkpoint
> > (tar jcf /tarballs/vserver1 /sys/namespaces/vserver1) and
> > certainly rename (mv /sys/namespaces/unnamed1 /sys/namespaces/sergeprivhome).
> >
> > One key observeration which I haven't made explicit is that you
> > never actually leave a nsid ("container"). If you start under
> > /vserver1, you will always be under /vserver1. I don't know of
> > any reason that would not be appropriate. If I start a nested
> > vserver from there, then to me it may be known as
> > 'vserver_testme', while to the admin of the machine, it would be
> > known as /vserver1/vserver_testme.
> >
> > This makes one possible implementation of the container struct:
> >
> > struct container {
> > struct container *parent;
> > char *name;
> > struct nsproxy *nsproxy;

> > struct list_head children;
> > };
> > struct nsproxy {
> > ...
> > struct container *container;
> > };
> >
> > Plus of course relevant sysfs stuff.
>
> I like the naming model. a few questions :
>
> how do you enter only a subset of namespaces of a nsproxy/container
> and not all of it ?

one container corresponds to one nsproxy which is one set of namespaces.
Are you asking how you would only switch your pid namespace but keep
your network namespace as the original?

I'm not sure that's something we want/need to support. (Can you cite a
use case?) But since I haven't specified how to ask for the nsproxy
switch anyway, it's too early to ask how we add a flag to specify a
namespace subset :)

I'm calling it a filesystem because we all understand the naming
semantics then, but that doesn't mean there'll be an actual
filesystem. It may all be hidden behind syscalls, in which case
we could do bind_ns(container_id, flags), where flags specifies which
namespaces to switch over.

But boy, then we need a new nsproxy to refcount those namespaces :)
Yuck.

> what flexibility the struct container is giving us ? why not have
> container == nsproxy ?

One reason is: once a process is in a container c1, if it unshares
and enters c2, it is still in c1. So if it was the last process out
of c1 to unshare, we need c1 to stick around, but the nsproxy will
be deleted. We could change the nsproxy semantics to match these
container semantics, but that makes the nsproxy implementation more
"magical".

And, of course, it's still under debate whether we'll keep the nsproxy.
This container model doesn't depend on the nsproxy, it's just exploiting
it. It can also work if all the namespaces are in the task_struct
explicitly.

> the recursivity model looks like extra overhead. it could be flat.

Absolutely not - it allows us to know with basically no accounting overhead which namespaces a process is a part of. It naturally fits our needs here. A process which is a part of /vserver1/calculate_pi is also a part of /vserver1, and so if we kill /vserver1, we should kill /vserver1/calculate_pi as well. Not doing this simple hierarchical model means we have to explicitly keep track of all the containers which a process is a part of.

As I said, once a process is in a container, it never leaves that container. It only enters additional ones. That model fits everyone's needs, without needing some funky API. If you consider the kinds of things users would need to specify otherwise - "Unshare, creating a new container, but leaving me in the old container", "unshare, creating a new container, and leaving the old container" (for which there is no need), and "unshare, keeping me inthe old container", the almost entirely implicit approach I just outlined is far far preferable IMO.

-serge